ADVANCED DRIVER ASSISTANCE SYSTEMS TECHNIQUES

USING A SINGLE CAMERA AND LIDAR

A Thesis
Presented to
The Academic Faculty

by

Sungwoo Han

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in the
School of Electrical and Computer Engineering

Georgia Institute of Technology

December 2018

ADVANCED DRIVER ASSISTANCE SYSTEMS TECHNIQUES

USING A SINGLE CAMERA AND LIDAR

Approved by:

Professor David Taylor, Advisor
School of Electrical and Computer Engineering
Georgia Institute of Technology

Professor Thomas Fuller
School of Chemical Engineering
Georgia Institute of Technology

Professor Michael Leamy
School of Mechanical Engineering
Georgia Institute of Technology

Date Approved: December 6, 2018

To my family and the Georgia Tech EcoCAR-3 Team.

ACKNOWLEDGEMENTS

This thesis would not have happened without the support, advice, and guidance of many people. I want to express my sincere appreciation to those who supported me throughout this thesis and taught me valuable things that I will be carrying as I move forward in my career.

I really appreciate Dr. David Taylor for continuously giving me the advice and guidance from the start to the end. I feel highly privileged to have him as my thesis advisor and learned so much from him. He helped me stay on the right track and gave thoughtful answers to all my concerns. I learned from him how to think and analyze problems with an engineering mind.

I also want to give great thanks to the EcoCAR-3 team, especially my teammates and the other team leaders for their support. Also, big thanks to Dr. Fuller and Dr. Leamy for their continuous support and encouragement as the EcoCAR-3 advisors.

A special thanks to both of my families in Korea and the United States. Without their support and encouragement, this thesis would not have been possible.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

The purpose of this thesis is to introduce the development and evaluation of some Advanced Driver Assistance Systems (ADAS) techniques using a single camera and LIDAR. This thesis describes the hardware architecture and software algorithms including lane mark detection (LMD), vehicle detection, vehicle tracking, longitudinal and lateral distance estimation, and sensor fusion. The intention of this thesis is to provide future ADAS developers with an informative technical reference.

CHAPTER 1

INTRODUCTION

With the rapid growth of Advanced Driver Assistance Systems (ADAS) technology, its market has become globally promising [1]. Most of the automotive companies and even some global IT companies have already jumped into the ADAS industry, and it is very common to see a vehicle with at least several ADAS features nowadays. It is indisputable that the ADAS technology stands in the future of the automotive industry.

As a part of Georgia Tech's participation in the EcoCAR-3 competition, I was tasked with leading the group responsible for designing, integrating and testing ADAS for our team's 2016 Chevy Camaro. This thesis documents the results of this effort and provides an informative technical reference for future ADAS developers.

The hardware architecture includes two single board computers, a camera, and LIDAR, and the specification and functionality of each component will be shared. The ADAS software includes the following features:

- Lane mark detection (LMD)
- Vehicle detection
- Vehicle tracking
- Longitudinal and lateral distance estimation
- Sensor fusion

The development method and result for each feature's algorithm is shared. The conclusions and future improvement ideas to develop a better ADAS are also provided in the last chapter.

CHAPTER 2

HARDWARE

2.1 System Overview

Figure 2.1 is a system overview diagram showing the hardware integration of the ADAS described in this thesis. The NVIDIA Jetson TX2 and NXP S32V234 are single-board computers for processing the ADAS algorithms. The Logitech C920 camera and LeddarTech LIDAR are the sensors for acquiring visual and existence information of target objects. The system communicates with vehicle controller of the subject vehicle in the form of a feedback loop.

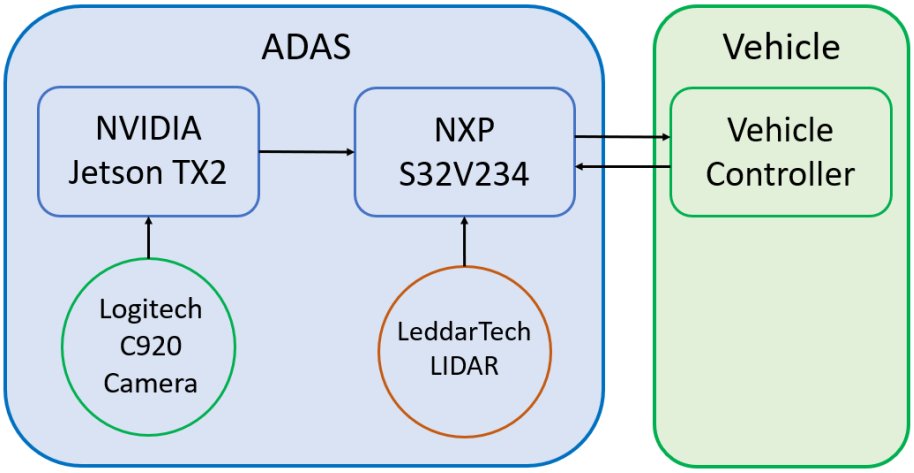

Figure 2.1. System Overview Diagram

2.2 Hardware Setup

Figure 2.2.1. Side (Left) and Aerial (Right) Views of Hardware Installation

The installation location for each hardware component is shown in the Figure 2.2.1. The Jetson TX2 and NXP S32V234 boards are installed in the trunk on the side. The Logitech camera is mounted inside the vehicle between the front windshield and back mirror. The LeddarTech LIDAR is mounted on the front bumper. Because the LIDAR uses light laser, its lens should not be blocked by any part of the bumper.

2.2.1 NVIDIA Jetson TX2 [2]

The Jetson TX2 is an embedded computer that is designed for applications such as computer vision and deep learning. The TX2 runs computationally expensive algorithms as fast as real-time by putting some or all loads on its graphics processing units (GPU). It also supports the NVIDIA Jetpack software development kit (SDK), which provides a wide range of libraries for deep learning, computer vision, GPU computing, and more. The TX2 runs any Linux operating system (OS) and provides various types of ports and peripherals including controller area network (CAN). NVIDIA also supports a strong community forum with questions and issues and provides plentiful technical resources and reference documents.

2.2.2 NXP S32V234 [3]

The NXP S32V234 is a vision processor designed to support various image processing applications, such as computer vision. It offers dual APEX-2 vision accelerator cores, embedded Image Sensor Processing (ISP), and a 3D Vivante GPU with OpenCL and OpenGL to accelerate image processing speed. It supports S32 Design Studio integrated development environment (IDE), which includes a compiler, debugger, Vision SDK, Linux board support package (BSP) and graph tools. Various ports and peripherals including CAN are also available.

2.2.3 Logitech C920 HD Pro Camera

The selected camera sensor is the Logitech C920 HD Pro. It can output up to 1920×1080 pixel resolution video at 30 frames per second. It provides a 78-degree horizontal field of view (FOV) as shown in the Figure 2.2.2. A stream of image frames captured by the camera is transferred to the TX2 to be processed by computer vision algorithms.

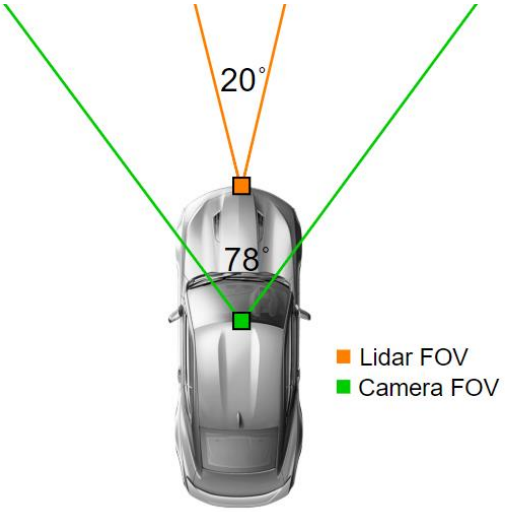

Figure 2.2.2. Camera and LIDAR Locations and Field of Views

2.2.4 LeddarTech LeddarVu-8 LIDAR

A LIDAR emits light waves, which are reflected off objects and travel back to the LIDAR receiver. It measures the round-trip time of the light to calculate the distance from the LIDAR to detected objects. LIDAR plays an important role in many ADAS applications, because it offers several advantages over a camera. Cameras have strengths in detecting and classifying objects, while LIDAR provides accurate distance measurements.

The LIDAR used in this thesis is the LeddarTech LeddarVu-8. The LeddarVu8 offers multi-object detection within 20 horizontal degrees by 0.3 vertical degrees fields of view (FOV) as shown in the Figure 2.2.2. The horizontal FOV is split over eight segments, and each segment provides distance measurement of a detected object.

CHAPTER 3

SOFTWARE

3.1 Software Architecture

The high-level system architecture is shown in Figure 3.1. The software algorithms used for ADAS include lane mark detection (LMD), vehicle detection, vehicle tracking, longitudinal and lateral distance estimation, and sensor fusion. The NVIDIA TX2 runs the computer vision algorithms and transfers the output data to the NXP S32V234. The S32V234 receives both computer vision and LIDAR data and integrates them into a final perception result.

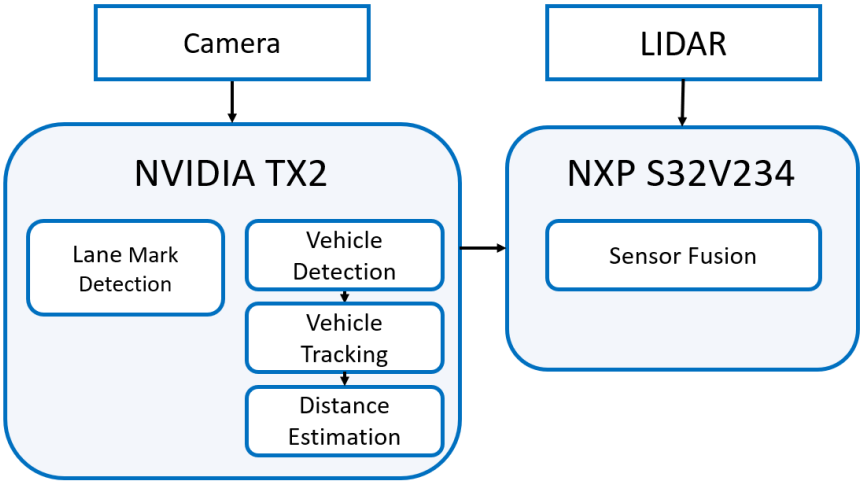

Figure 3.1. System Architecture

3.2 Lane Mark Detection (LMD)

LMD takes a stream of images as inputs to detect left and right marks of the lane on which the subject vehicle is driving. The LMD output can be useful in many ADAS applications such as auto steering and adaptive cruise control. In auto steering, the LMD output is used to control the subject vehicle to stay within the lane while driving. In adaptive cruise control, the result can be useful in identifying whether the detected car in the front is driving in the same lane as the subject vehicle or in an adjacent lane. LMD is an essential feature for ADAS.

3.2.1 Grayscale Conversion

Prior to searching for lane marks, the captured image is first converted to a grayscale image. The purpose of this step is to reduce the size of the image data from a three-layer color matrix to a single-layer grayscale matrix. A grayscale pixel is a weighted average of red, green, and blue (RGB) pixel values. The formula to convert a RGB pixel to a grayscale pixel is following [4]:

$$\text{Grayscale Pixel} = 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B$$

The input RGB image and the output grayscale image are shown in the Figure 3.2.1.

Figure 3.2.1. The Input RGB (Left) and Grayscale (Right) Images

3.2.2 Gaussian Blur [5]

The grayscale image is blurred by Gaussian blur to reduce the image noise and details. Blurring the image helps to minimize undesired edge detection. A Gaussian kernel is a square matrix of pixels where the values correspond to the values of Gaussian curve and is shown in the Figure 3.2.2.

$$\frac{1}{273}\begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}$$

Figure 3.2.2. 5 × 5 Gaussian Blur Kernel [5]

Each pixel in the input image is convolved with the Gaussian kernel as shown in the Figure 3.2.3. The center of the kernel is placed on the input image pixel, and the

overlapping pixels are multiplied by the corresponding kernel values. Then, the multiplied values are added and assigned to the output image pixel.

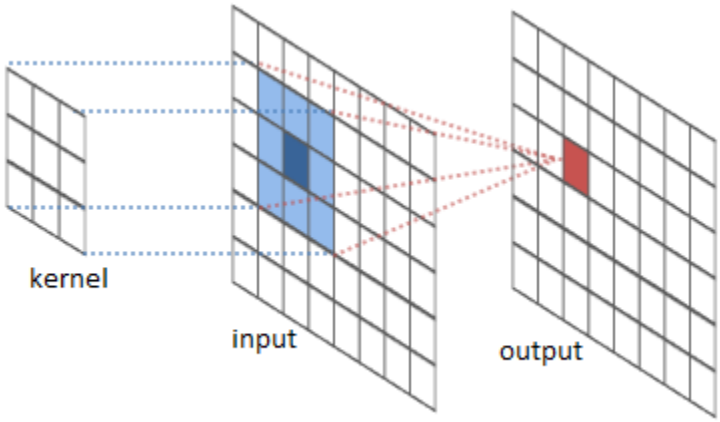

Figure 3.2.3. Convolution of Input Image with a Gaussian Kernel [6]

The input grayscale image and output blurred image are shown in the Figure 3.2.4.

Figure 3.2.4. The Input Grayscale (Left) and Blurred (Right) Images

3.2.3 Edge Detection [7]

Edges on the image can be detected by an edge detector operator that measures the pixel intensity gradient of the image. A high gradient magnitude indicates that the intensity of the pixels is changing rapidly, implying an edge. The edges are always

normal to the direction of gradient since the intensity does not change along an edge, but across the edge. A Sobel kernel is used to calculate the first derivative of pixel intensity in horizontal (G_x) and vertical (G_y) directions. Assuming I is the input image, G_x and G_y are found as follows:

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} \cdot I \qquad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} \cdot I$$

Then, the edge gradient and direction of each pixel are found as follows:

$$\text{Gradient } (G) = \sqrt{G_x^2 + G_y^2}$$

$$\text{Angle } (\theta) = \tan^{-1}\left(\frac{G_y}{G_x}\right)$$

After calculating the gradient and direction for every pixel in the image, the local gradient maximum is found in the direction of the gradient. As shown in the Figure 3.2.5, when A, B, and C are in the gradient direction and pixel A has the maximum gradient, A is considered as a candidate of the edges, and B and C are suppressed to zero. This step is also called non-maximum suppression.

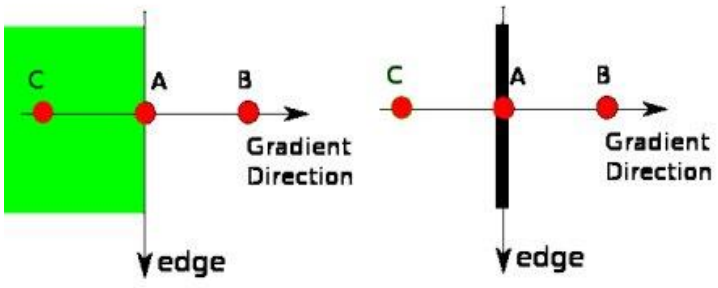

Figure 3.2.5. Non-maximum Suppression [7]

Finally, the candidate edges found in the previous step are filtered by hysteresis thresholding, minVal and maxVal. Any edges below the minVal are discarded. Any edges above the maxVal are considered as 'sure-edge'. The edges lying in between the maxVal and minVal are determined based on their connection with the sure-edge. As shown in the Figure 3.2.6, C is considered as a part of edge because it is connected to A, which is sure-edge. But, even though B is in the same region as C, it is not considered as an edge because B is not part of any sure-edge. The output image is a binary matrix with ones (white) representing the edge and zeros (black) representing the rest.

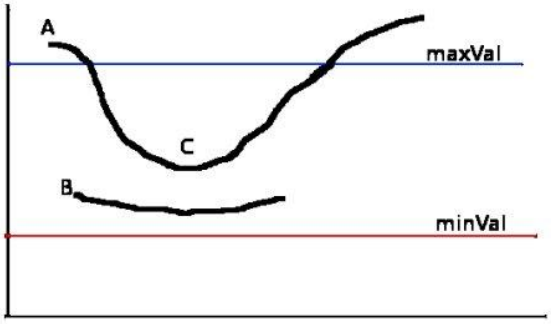

Figure 3.2.6. Hysteresis Thresholding with MinVal and MaxVal [7]

The input blurred image and output edge image are shown in the Figure 3.2.7.

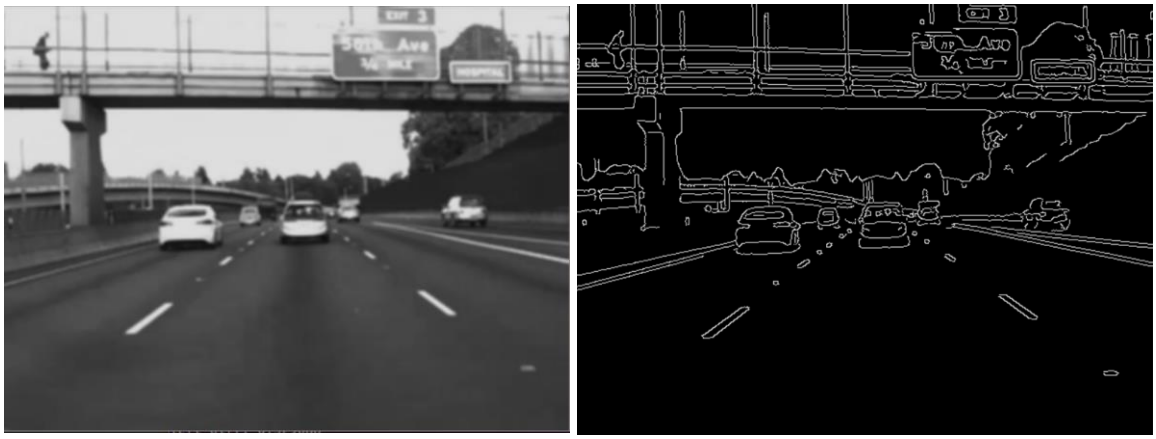

Figure 3.2.7. The Input Blurred (Left) and Output Edge (Right) Images

3.2.4 Hough Line Transform [8]

Straight lines are detected on a binary edge image using a technique called Hough line transform. A line can be expressed in a polar coordinate system using two variables (θ, r) as shown in the Figure 3.2.8.

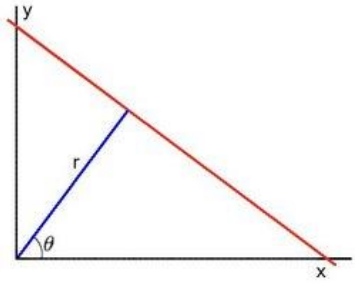

Figure 3.2.8. A Line Expressed in Polar Coordinate System [8]

A line equation can be written as:

$$y = \left(-\frac{\cos\theta}{\sin\theta}\right)x + \left(\frac{r}{\sin\theta}\right)$$

which can be rearranged as:

$$r = x \cdot \cos\theta + y \cdot \sin\theta$$

A family of lines that go through a single point can be found by varying θ and r, and a corresponding sinusoid can be plotted on a θ- r plane as shown in the Figure 3.2.9.

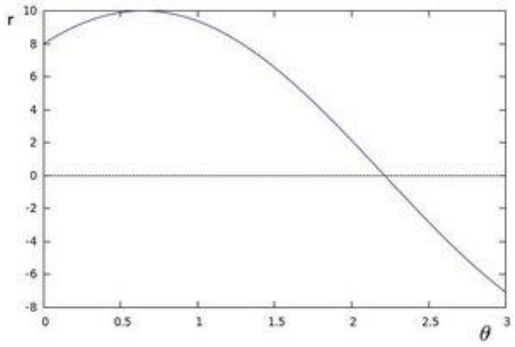

Figure 3.2.9. A Family of Lines for a Single Point [8]

If two or more sinusoids intersect at one point, it means that the corresponding points are on the same line, and those sinusoids can be plotted as shown in the Figure 3.2.10.

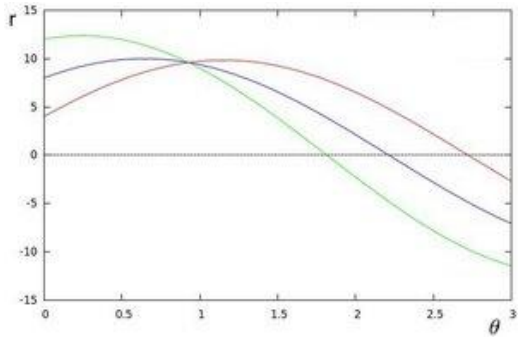

Figure 3.2.10. Three Sinusoids Representing Three Points Lying on the Same Line [8]

The number of sinusoids intersecting at a point can be a threshold variable for detecting lines. If a point has a number of intersections above a threshold, it is declared as a line with the corresponding parameters (θ, r) of the intersection point. The input edge image and output with red straight lines are shown in the Figure 3.2.11.

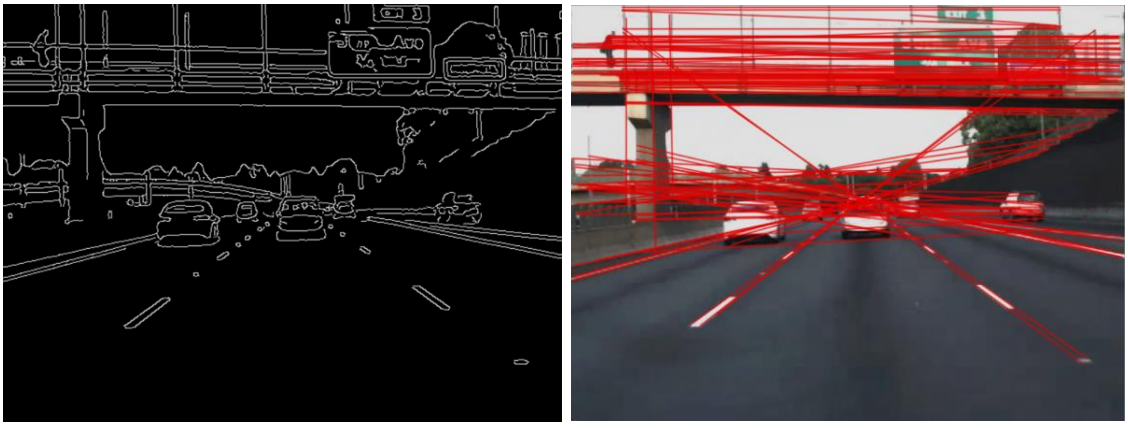

Figure 3.2.11. The Input Edge Image (Left) and Output with Red Straight Lines (Right)

3.2.5 Additional Filtering

After the Hough line transform, all straight lines in the image are found as candidates of the true left and right lane marks of the host lane. As shown in the Figure 3.2.11, there can be too many candidates. Additional filters can be used to extract the true lane marks as follows:

- **Region of Interest (ROI)**

  Many candidates can be filtered by using a ROI before converting the RGB image to grayscale. The camera is installed at an angle so that the true lane marks are most likely to be on the bottom half region of the image frame unless the vehicle is driving on an extreme vertical curve road. Therefore, the bottom half region is selected as the ROI which indicates that the LMD only searches lines inside the ROI instead of the entire frame as shown in the Figure 3.2.12. This step also reduces the computation load by using a smaller image matrix for the image processing and filtering.

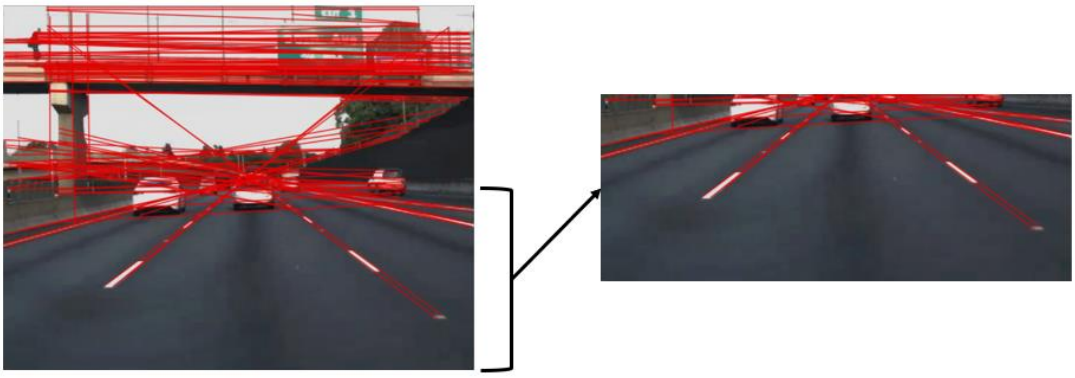

Figure 3.2.12. Region of Interest

- **Left and Right Lane Mark Candidates**

  The lane mark candidates can be divided into two categories depending on their

line angles: candidates for left lane mark and candidates for right lane mark. As shown in the Figure 3.2.13, given that an angle of a line is measured from a positive x-axis going counter-clock wise, the angle of a right lane mark is most likely between 90 and 180 degrees unless the subject vehicle is changing lanes or driving on a sharp curve. Lines that have such angle are right lane mark candidates. Using the same logic, the angle of a left lane mark is most likely between 0 and 90 degrees. Lines that have such angle are left lane candidates.

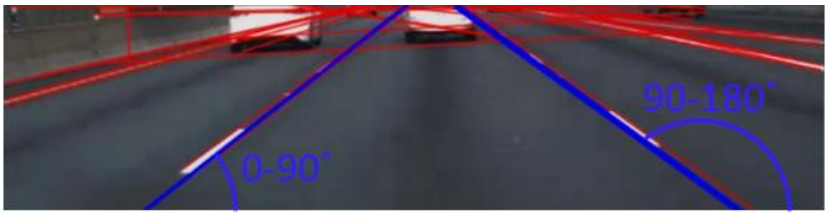

Figure 3.2.13. Line Angles

- **Filtering by Position**

  The lane mark candidates of each category can be filtered by line position. The position of a mid-point of each line is calculated. Two mid-points are shown by green dots as an example in the Figure 3.2.14. If a line is a left lane mark candidate, but its mid-point is located on the right half plane of the image, it is removed from the candidate list. Using the same logic, if a mid-point of a right lane candidate is located on the left half plane of the image, it is removed.

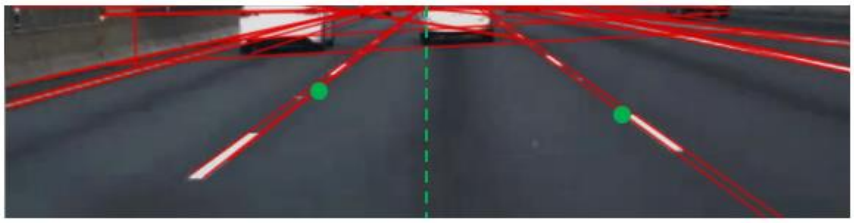

Figure 3.2.14. Positions of Line Mid-points

- **Filtering by Angle**

  Finally, lane mark candidates are filtered by their angles until only one remains for the left and right categories. The line whose angle is the closest to 90 degrees is selected as the true lane mark for each category. As shown in the Figure 3.2.13, the true left and right lane marks are the most vertical lines that have their angles closest to 90 degrees.

After the filtering processes, the LMD outputs one out of three possible options: two lane marks (left and right), one lane mark (left or right), or none. The LMD flow chart diagram is shown in the Figure 3.2.15.

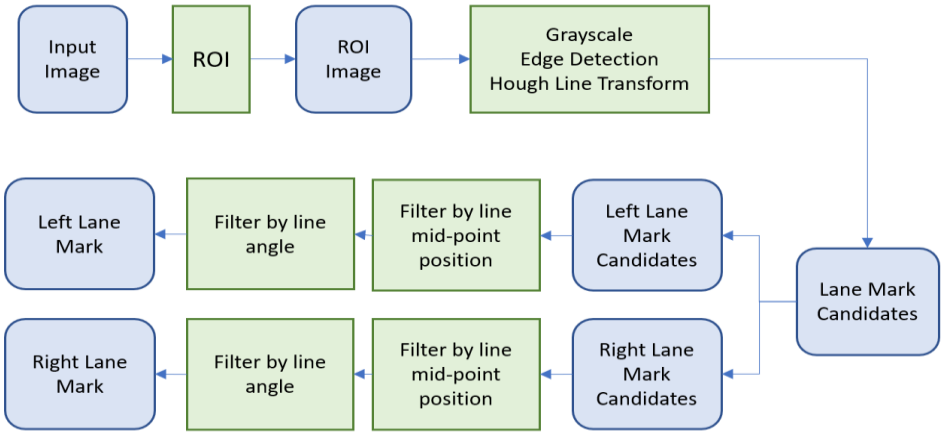

Figure 3.2.15. Flow Chart Diagram of Lane Mark Detection

3.2.6. Results

Desired Results | Undesired Results

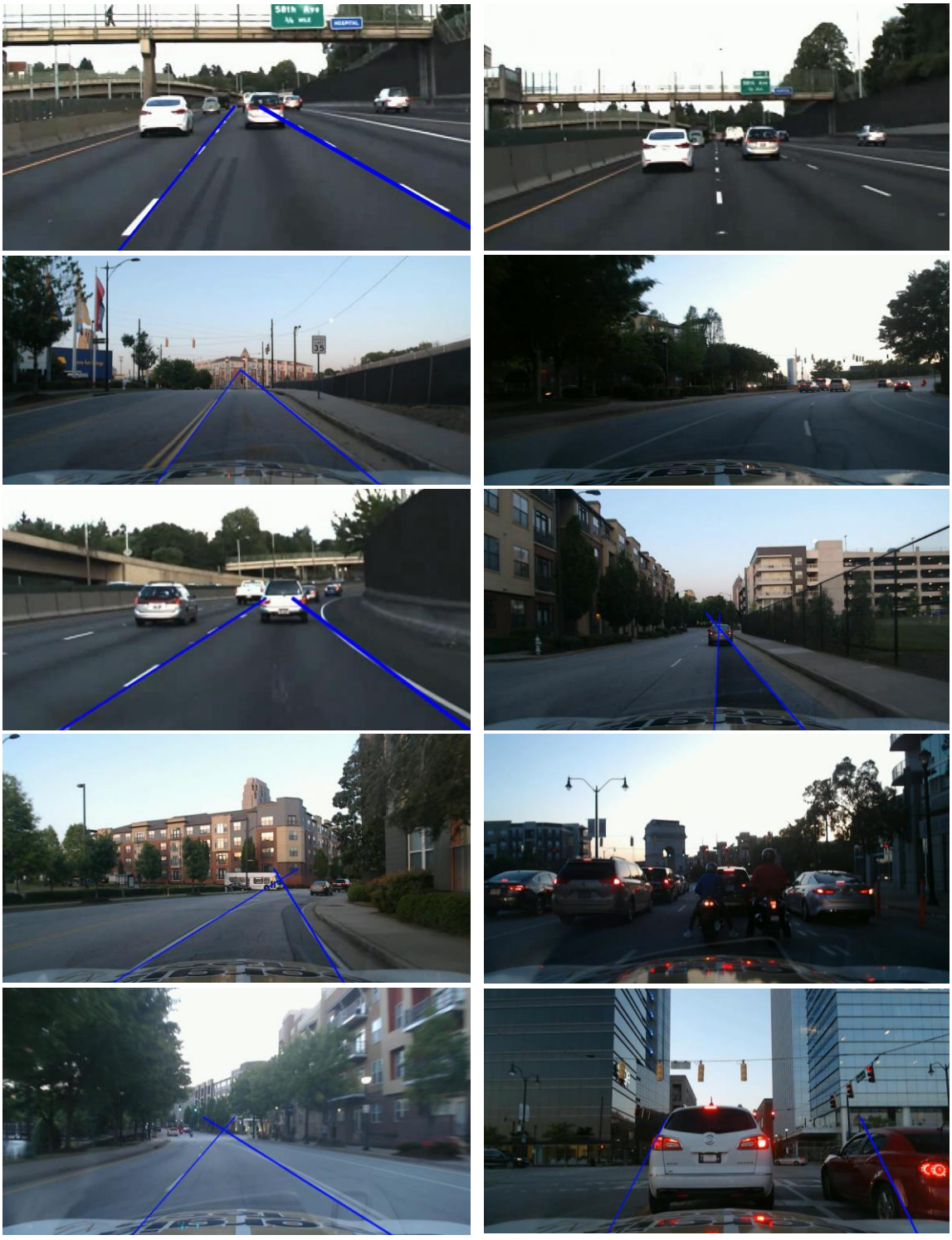

Figure 3.2.16. Desired and Undesired Results of Lane Mark Detection

The Figure 3.2.16 above shows the results of the LMD. When the lane marks are clearly visible, the LMD detects lane marks well on most straight and wide curve roads. But, it sometimes misses lane marks when the subject vehicle is driving on a sharp curve or is changing lanes. Also, the LMD have false detections when the pavement is not uniform, or some portion of the lane marks is occluded. Sometimes, LMD falsely recognizes the outline of a vehicle and/or other objects as a lane mark.

3.3 Vehicle Detection

Vehicle detection is a type of object detection, a technology that detects a specific target and locates its position in a pixel image. Nowadays, it is very common to find various applications of object detection, such as face detection for security, product detection in the manufacturing industry, human detection for people counting, and so on. ADAS also uses object detection to detect various objects such as vehicles, pedestrians, and traffic signs. Deep learning and convolutional neural network (CNN) have become dominant for object detection due to the recent advance of processing capacity of hardware technology. CNN is a type multi-layer artificial neural network that was inspired by the neural system of animal visual cortex. CNN is a fundamental technology for many state of the art object detection systems including faster region based convolutional neural networks (Faster R-CNN) [9], region based fully convolutional network (R-FCN) [10], single shot multi-box detection (SSD) [11], and you only look once (YOLO) [12] systems. The vehicle detection algorithm used in this thesis uses YOLO version 2 (YOLOv2) [13] which detects the rear of vehicles moving in the same direction as the subject vehicle.

3.3.1. How YOLO Works [12]

Like other CNN based detection systems, YOLO applies CNN on input images to detect its target objects. But, YOLO differentiates itself from other systems by running CNN only once, which accounts for its name. Prior to detecting objects, YOLO divides the input image into a grid of 13 by 13 cells. Each cell is responsible for predicting five bounding boxes, for a total of 845 boxes. A bounding box describes a rectangle that

encloses any kind of object. YOLO measures a confidence score indicating how certain it is that the predicted bounding box holds an object. Each bounding box also measures a classification score to predict a class with a probability distribution over all the possible classes. In this case, there is only one possible class, a vehicle. The confidence and classification scores are combined into a single final score. The bounding boxes with low scores are filtered by a threshold and only the meaningful boxes remain as shown in the Figure 3.3.1. YOLO handles this process of finding bounding box coordinates and scores from input images as a single regression problem. Therefore, YOLO runs the CNN only once, which makes it one of the fastest neural network systems for vehicle detection.

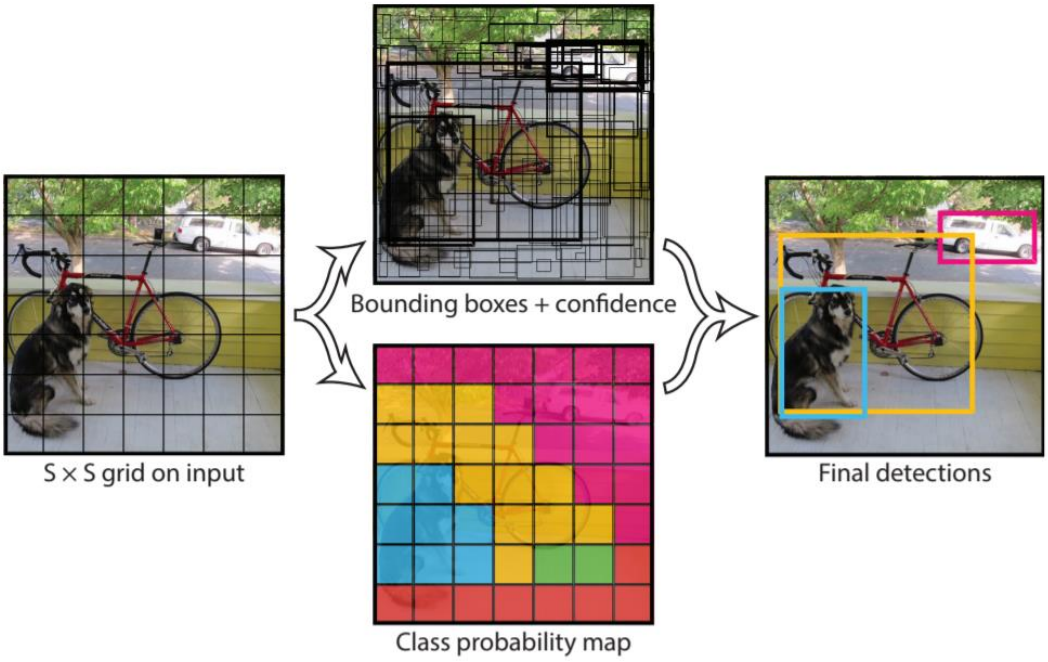

Figure 3.3.1 High-Level Flow Chart of You Only Look Once [12]

3.3.2 Training a Vehicle Detector

For training a YOLO detector, Joseph Redmond proposed a CNN model called Darknet-19 [13]. Redmond provided convolutional weights that are pre-trained on a well-known image database called ImageNet using the Darknet-19 model. The detailed layers of Darknet-19 are shown in the Figure 3.3.2.

Type	Filters	Size/Stride	Output
Convolutional	32	3×3	224×224
Maxpool		$2 \times 2/2$	112×112
Convolutional	64	3×3	112×112
Maxpool		$2 \times 2/2$	56×56
Convolutional	128	3×3	56×56
Convolutional	64	1×1	56×56
Convolutional	128	3×3	56×56
Maxpool		$2 \times 2/2$	28×28
Convolutional	256	3×3	28×28
Convolutional	128	1×1	28×28
Convolutional	256	3×3	28×28
Maxpool		$2 \times 2/2$	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Maxpool		$2 \times 2/2$	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	1000	1×1	7×7
Avgpool		Global	1000
Softmax			

Figure 3.3.2. Darknet-19 by Joseph Redmon [13]

Using the convolutional weights, a vehicle detector is trained on a custom dataset. The custom dataset includes vehicle rear images from common objects in context (COCO) dataset [14]. To add more variations and sizes of vehicles, many pictures are captured using the Logitech camera and added to the custom dataset. Each vehicle rear is enclosed by a bounding box and is labelled as ‘vehicle’. The bounding box is used to locate the position of the vehicle in the image during the training process.

3.3.3 Evaluation Method

To evaluate the performance of the vehicle detection, a test video is recorded with the Logitech camera installed in the subject vehicle. The test video covers some common driving scenarios, such as following a vehicle in the same lane, following a vehicle in the adjacent lane, passing a vehicle, and being passed by a vehicle. The vehicle detection is implemented on the test video to log outputs including frame numbers and bounding boxes of the detected vehicles. On the same video, ground truth data is also created by manually tagging the vehicles with bounding boxes using the MATLAB ground truth labeling tool. The vehicle detection algorithm output and the ground truth data are visualized in Figure 3.3.3.

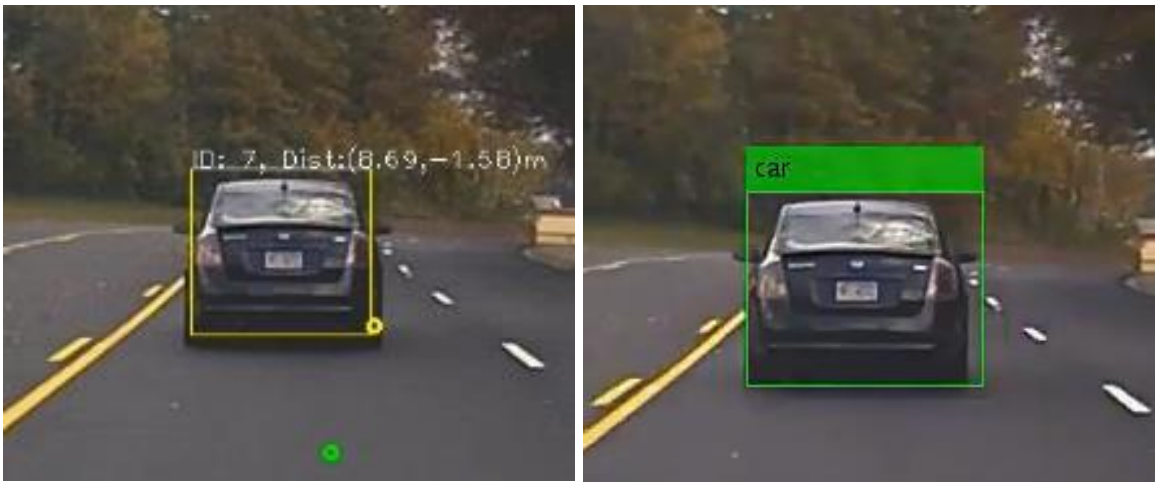

Figure 3.3.3. Vehicle Detection Output (Left) and Ground Truth Data (Right)

The performance of the vehicle detection is evaluated by comparing its output with the ground truth data using the following metrics:

- True positive rate (TPR): True positive (TP) indicates that the algorithm correctly identifies the vehicle presence. TPR measures the sensitivity of the algorithm and is calculated by dividing TP by real positive or ground truth positive (P), which is a sum of TP and false negative (FN):

$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{P}} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}$$

- False positive rate (FPR): False positive (FP) indicates that the algorithm falsely identifies a non-existent vehicle. FNR is calculated by dividing FP by the real negative or ground truth negative (N), which is a sum of FP and true negative (TN):

$$\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{N}} = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TN}}$$

- True negative rate (TNR): True negative (TN) indicates that the algorithm correctly identifies that there is no vehicle. TNR measures the specificity or selectivity of the algorithm and is calculated by dividing TN by N:

$$\mathrm{TNR} = \frac{\mathrm{TN}}{\mathrm{N}} = \frac{\mathrm{TN}}{\mathrm{FP} + \mathrm{TN}}$$

- False negative rate (FNR): False negative (FN) indicates that the algorithm fails to detect an existing vehicle. FNR measures the miss rate of the algorithm and is calculated by dividing FN by P:

$$\mathrm{FNR} = \frac{\mathrm{FN}}{\mathrm{P}} = \frac{\mathrm{FN}}{\mathrm{TP} + \mathrm{FN}}$$

- Precision: Precision or positive predictive value is calculated as:

$$\mathrm{Precision} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}}$$

- Accuracy: Accuracy is calculated as:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{P} + \text{N}} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

For each vehicle detection, intersection over union (IoU) between the areas of the detection and ground truth bounding boxes is calculated. When the IoU of a detection is over 70 percent, it is considered as a correct detection.

3.3.4. Results

The result of the detection algorithm is shown in Figure 3.3.4 and Table 3.3.1. The algorithm TPR and FNR show 83.36% and 16.64%, respectively. The vehicle detection has high TPR when the target vehicles are close to the subject vehicle, but the TPR starts to decrease when the targets are farther away. Sometimes, the algorithm falsely detects oncoming vehicles as targets. The FPR and TNR are N/A because the recorded video has no true negative which means that it includes at least one vehicle in every frame. Precision and accuracy of the algorithm are 95.82% and 80.43%, respectively.

Metrics	Vehicle Detection	Ground Truth
TP	481	577
FP	21	0
TN	0	0
FN	96	0
TPR	83.36%	100%
FPR	N/A	N/A
TNR	N/A	N/A
FNR	16.64%	0%
Precision	95.82%	100%
Accuracy	80.43%	100%

Table 3.3.1. Vehicle Detection and Ground Truth Result

Desired Results Undesired Results

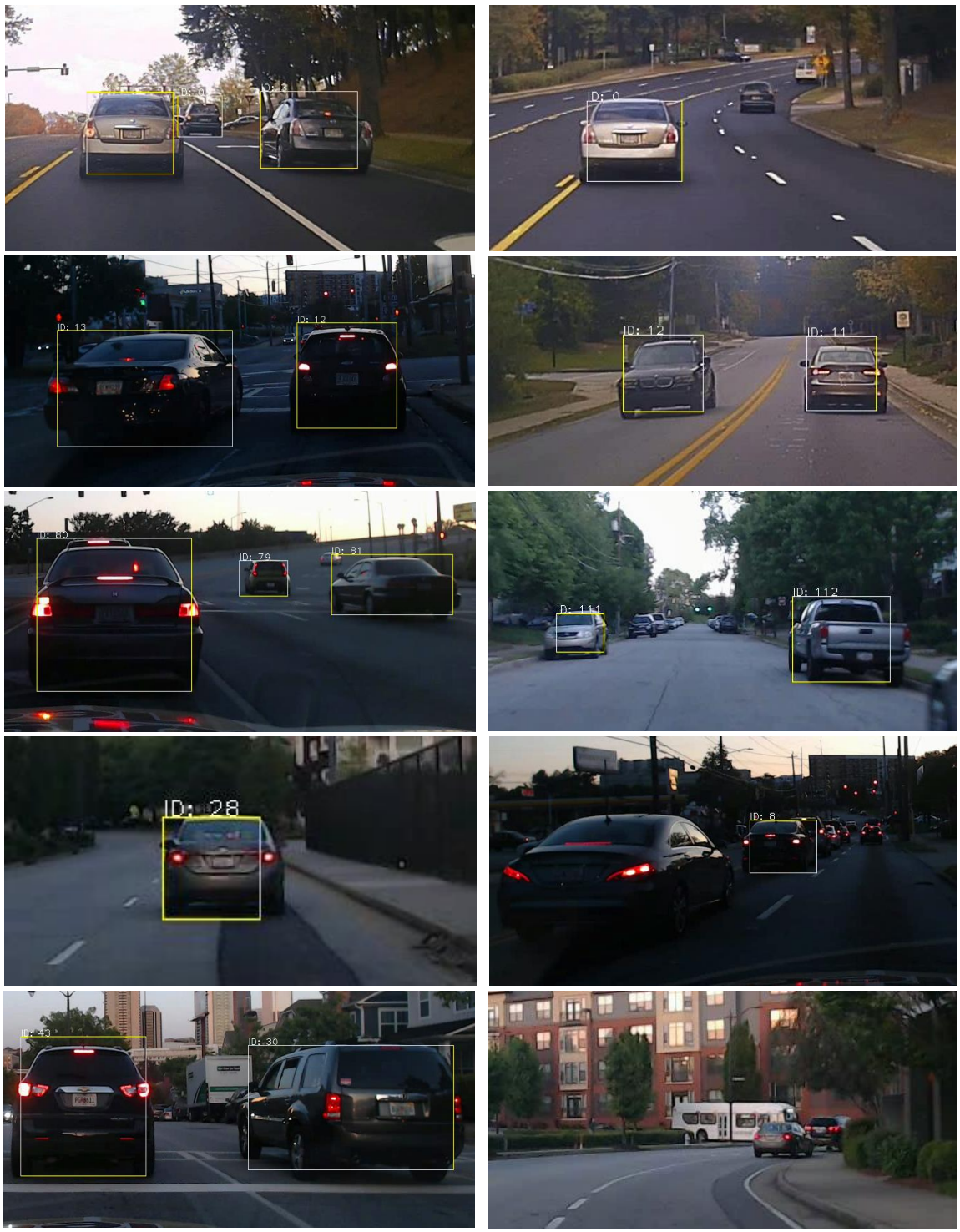

Figure 3.3.4. Desired and Undesired Results of Vehicle Detection

3.4. Vehicle Tracking

Object tracking provides a robust way to track detections through consecutive frames. The vehicle tracking algorithm detects the motion of a target vehicle and predicts its next location in the frame. Tracking can also provide a way to filter out unreliable detections by keeping track of each target vehicle.

3.4.1 Kalman Filter

The tracking algorithm, as implemented, relies heavily on a technology called Kalman filter [15]. The Kalman filter is a set of mathematical equations to calculate the present and future states of a system. The filter works as a form of feedback control. It projects the current state of the system and error covariance estimate to predict the future state in time. Then, it takes a feedback in the form of a new noisy measurement to update or improve the predicted state. This process can be summarized as a two-step feedback loop as shown in Figure 3.4.1. In the case of vehicle tracking, the system takes the velocity of a moving vehicle to predict its future location.

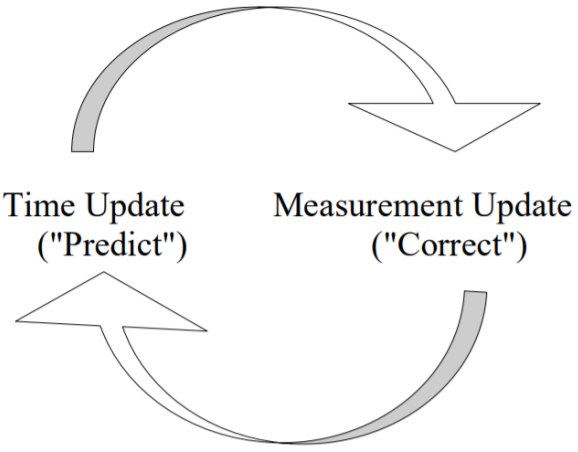

Figure 3.4.1. Discrete Kalman Filter Cycle [16]

This two-step process can also be expressed with two equations [17] as:

$$\text{Prediction Equation:} \qquad X(n) = F \cdot X(n-1) + V_q(n-1)$$

$$\text{Update Equation:} \qquad Y(n) = H \cdot X(n) + V_p(n)$$

where $X(n)$ is the estimated state variable, and $Y(n)$ is the measurement variable. F is the state transition matrix, and H is the measurement matrix. $V_q(n-1)$ and $V_p(n)$ denote the system noise and measurement noise, respectively. The two equations can be rewritten in matrix forms as:

$$X(n) = \begin{bmatrix} x_t \\ y_t \\ \dot{x}_t \\ \dot{y}_t \end{bmatrix} = \begin{bmatrix} 1 & 0 & T & 0 \\ 0 & 1 & 0 & T \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{t-1} \\ y_{t-1} \\ \dot{x}_{t-1} \\ \dot{y}_{t-1} \end{bmatrix} + V_q(n-1)$$

$$Y(n) = \begin{bmatrix} x_t \\ y_t \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_t \\ y_t \\ \dot{x}_t \\ \dot{y}_t \end{bmatrix} + V_p(n)$$

where (x_t, y_t) and $(\dot{x}_t, \dot{y}_t)$ are the position and velocity state variables of a detected vehicle in a 2-D system, respectively. T represents the time interval between consecutive frames.

3.4.2 Hungarian Algorithm

The Hungarian algorithm is implemented for tracking multiple objects. The Hungarian algorithm is a combinatorial optimization technique used for optimal cost-based assignment. When a new detection is present, a track is assigned to the detection. A track contains information about the detection, such as a unique ID, current state, Kalman filter's predicted state, age, visible, and invisible count. The cost is defined as the

distance between the center of the new detection bounding box and the Kalman filter's predicted state coordinates of an existing track. For each detection, the cost for all existing tracks are calculated. The detection is assigned to the track with the minimum cost.

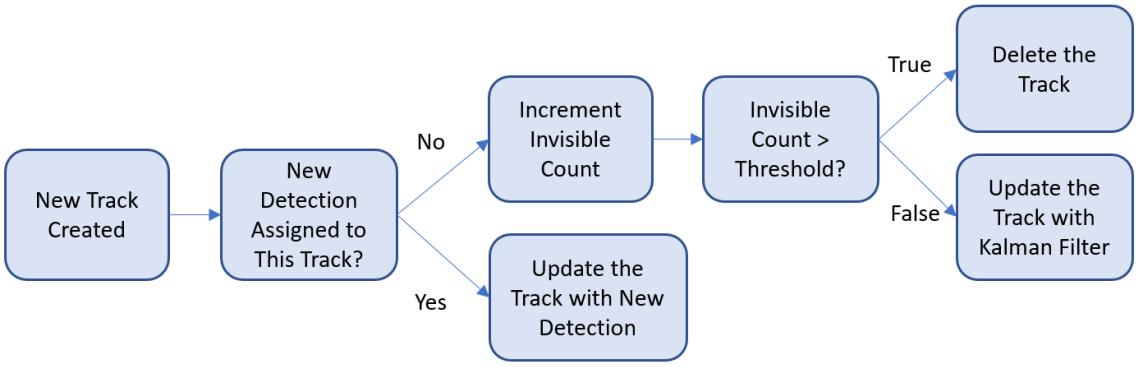

Figure 3.4.2. The Flow Chart Diagram of Multiple Tracking using Tracks

Figure 3.4.2 shows the flow chart diagram of how the vehicle tracking handles multiple tracking using tracks. When a vehicle is detected with no assigned track, a new track is created. In the next frame, if a new detection is assigned to this track by the Hungarian algorithm assignment, the track is updated. If none of the new detections were assigned to this track, the track is updated based on the Kalman filter's prediction state. The track is deleted if there are no assignments for a threshold number of consecutive frames.

The nature of such a state tracker allows for an accurate prediction and correction to smooth out the frames of a detection. Using the Kalman filter, the system can track a target vehicle even when the vehicle is completely occluded for a few frames by some moving object.

3.4.3. Results

Figure 3.4.3. Two Examples of Vehicle Tracking Using Kalman Filter

Figure 3.4.3 shows two examples of vehicle tracking. Using the predicted states of the Kalman filter, the detected vehicle is still tracked even when it is completely occluded by another vehicle. The unique ID assigned to each detected vehicle stays with the correct detection even when two detected vehicles are overlapped with each other brifely and break apart.

3.5 Longitudinal and Lateral Distance Estimation with Sensor Fusion

When a target vehicle is detected, its longitudinal and lateral position relative to the subject vehicle is measured and assigned to the detection. The distances are measured by two sources: camera and LIDAR. A sensor fusion algorithm is used to integrate the two measurements of the same object into a final output.

3.5.1 Camera Based Distance Estimation

When the vehicle detection algorithm detects a vehicle, its relative position from the camera is calculated. This step involves converting the 2-D pixel image coordinate system to the 3-D real world coordinate system. Figure 3.5.1 shows the coordinate system conversion using a pinhole camera model.

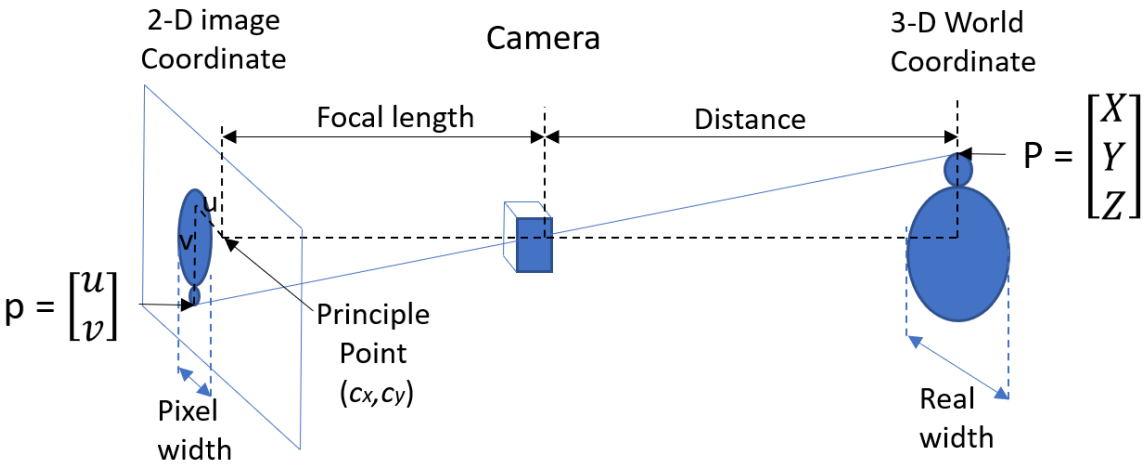

Figure 3.5.1. Coordinate System Conversion from 2-D to 3-D using a Pinhole Camera Model

3.5.1.1. Longitudinal Distance Estimation

The longitudinal distance is estimated using the width of the detection bounding box because the width variation of a vehicle is usually limited by the lane width, whereas the height variation can be large depending on the type of a vehicle. The longitudinal distance is calculated using the following equation:

$$\text{Longitudinal Distance} = \frac{\text{Vehicle Width} \cdot \text{Focal Length}}{\text{Width of the Bounding Box}}$$

Since the actual width of a detected vehicle is unknown, the estimated average of a vehicle is used for the vehicle width. The focal length is one of the intrinsic parameters of the Logitech C920 camera.

3.5.1.2. Lateral Distance Estimation

Using the calculated longitudinal distance, the lateral distance can also be calculated with the ideal pinhole camera model matrix equation as following:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

u and v indicate the pixel coordinates on the image frame as shown in the Figure 3.5.1. c_x and c_y indicate the coordinates of the camera's principle point. f is the focal length of the camera. X, Y, and Z are the coordinates of the center of the target vehicle bounding box in the 3-D world coordinates. X is the lateral position and Y is the longitudinal position.

3.5.2 Sensor Fusion

Sensor fusion is a technology that integrates or fuses data from multiple sensors to develop a more reliable and accurate system. This technology is used in many applications nowadays including robotics, defense systems, and autonomous driving. One of the most cited sensor fusion models is the one developed by the U.S defense research foundation, Joint Directors of Laboratory (JDL) [18]. The JDL introduces different levels of sensor fusion models based on the combination of level of data abstraction and level of problem-space complexity. These levels are defined as follows:

- **Level 0:** Estimation of States of Sub-Object Entities (e.g. signals, features)
- **Level 1:** Estimation of States of Discrete Physical Objects (e.g. vehicles, buildings)
- **Level 2:** Estimation of Relationships Among Entities (e.g. aggregates, cuing, intent, acting on)
- **Level 3:** Estimation of Impacts (e.g. consequences of threat activities on one's own assets and goals)

The Level 1 model approach is used to combine detections from the camera and LIDAR. The two sensors operate simultaneously to detect the states of objects, which are then integrated into a single final data.

ADAS gains a huge advantage using sensor fusion. LIDAR has its strength in measuring accurate and reliable relative distance of a detected object but has a weakness in classifying the type of object. On the other hand, the camera can classify the type of detected object but does not always provide accurate relative distance. Combining the

two sensors with a good sensor fusion algorithm allows ADAS to take advantage of each sensor's strength.

3.5.2.1. Coordinate Calibration

Before fusing the camera and LIDAR detections, their coordinate systems need to be calibrated since the mounting locations of the LIDAR and camera are different as shown in the Figure 2.2.1. When viewed from the front, both sensors are mounted in the middle of the car, so no lateral offset is needed. However, longitudinal offset is needed because the LIDAR is mounted on the front bumper whereas the camera is mounted behind the front wind shield. About one meter of longitudinal offset is added to the camera detections. After the coordinate systems are calibrated, the camera and LIDAR detections as well as the sensor fusion outputs are plotted on a bird's eye plot as shown in the Figure 3.5.2. The center of the front bumper of the subject car is on the origin of the plot.

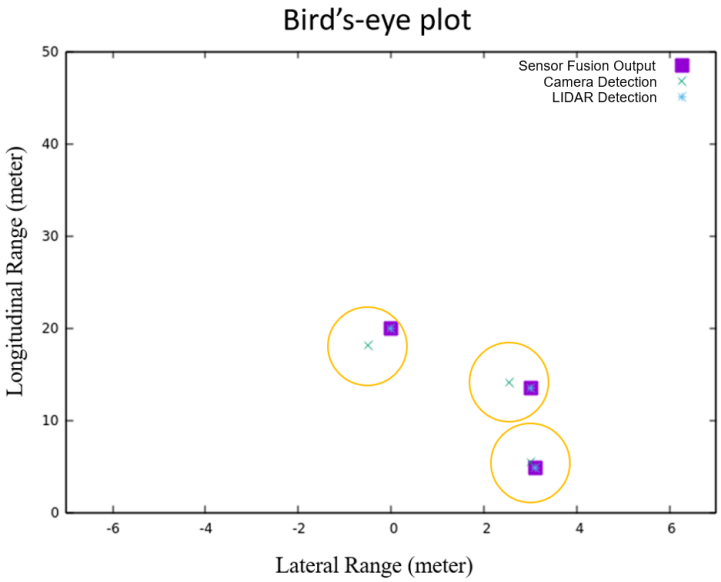

Figure 3.5.2. Camera, LIDAR, and Sensor Fusion Points on Bird's Eye Plot

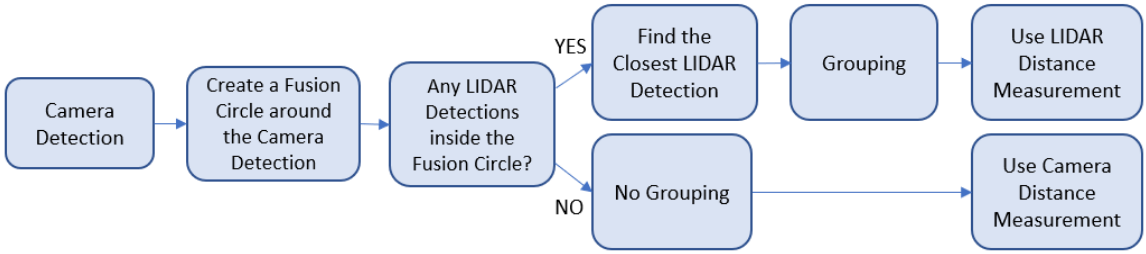

Figure 3.5.3. Flow Chart Diagram of Sensor Fusion Algorithm

3.5.2.2. Grouping Camera and LIDAR Detections

Figure 3.5.3 is a flow chart diagram of the sensor fusion algorithm. The camera-based vehicle detection detects a vehicle and estimates its longitudinal and lateral coordinates. A virtual fusion circle with a threshold radius is created around this detection coordinates as shown by the orange circle in Figure 3.5.2. The virtual fusion circle is used to check if any LIDAR detections are close enough to a camera detection for sensor fusion. The system checks if any of new LIDAR detections fall inside this orange circle. If a single LIDAR detection is inside of the circle as shown in Figure 3.5.2, the camera and LIDAR detections are grouped to be a single fusion detection. This grouping indicates that the distance between the LIDAR and camera detections is smaller than the threshold distance (radius of fusion circle) and the two detections are detecting the same object. In this case, the LIDAR's distance measurement is used for the final output because the LIDAR is more reliable in measuring distance. If multiple LIDAR detections are inside the fusion circle, the one that is closest to the camera detection is grouped. If no LIDAR detection is inside the fusion circle, no grouping is performed. In this case, the system uses the camera distance measurement as the final output.

3.5.2.3. Time Synchronization with Multi-Threads

Computing speed of the camera-based vehicle detection and LIDAR detection are different. The camera-based system outputs detection results at about 5 Hertz, whereas the LIDAR outputs detection results at about 20 Hertz. A time synchronization of the two detection systems is required for the sensor fusion. To handle this, three threads are created: a camera detection thread, a LIDAR detection thread, and a sensor fusion thread. The steps taken to achieve the time synchronization are shown in Figure 3.5.4.

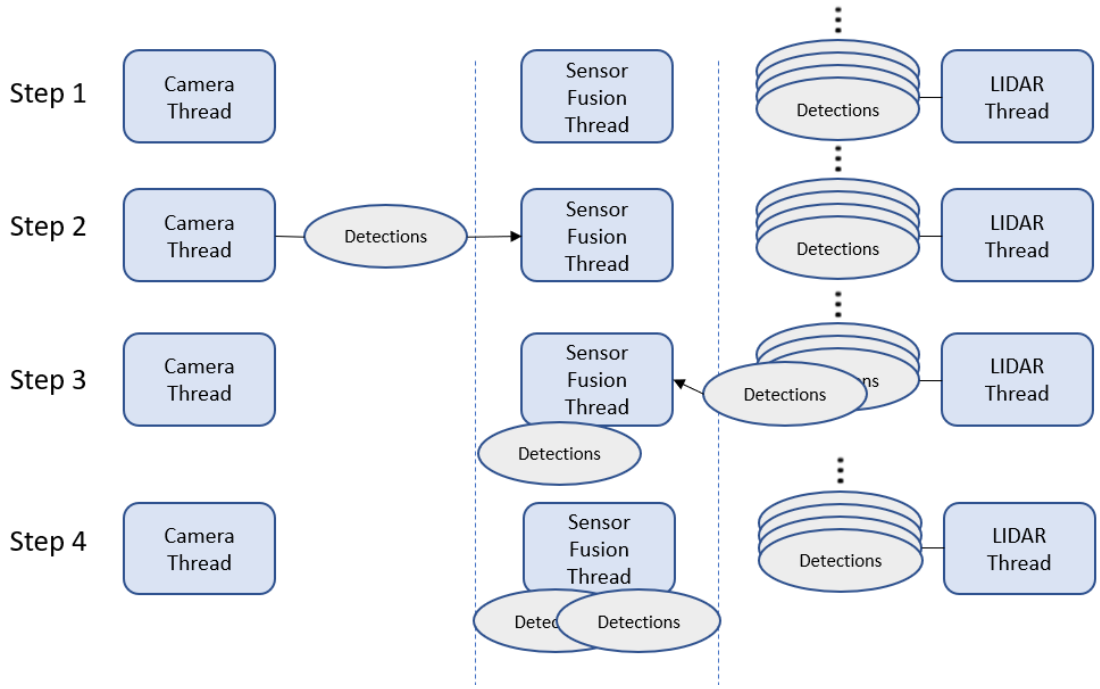

Figure 3.5.4. Time Synchronization with Multi-Threads

- **Step 1:** The sensor fusion thread monitors only the camera detection thread for a new detection output and ignores detections from the LIDAR thread.

- **Step 2:** The sensor fusion thread receives a detection output from the camera thread.
- **Step 3:** The sensor fusion thread monitors the LIDAR thread and receives the next available LIDAR detection.
- **Step 4:** The sensor fusion thread combines the two detections from each thread and outputs a final vehicle detection result.

3.5.3. Evaluation Method

To evaluate the accuracy of the longitudinal distance estimation, a test track is set up with traffic cones in a controlled parking lot. A target vehicle is parked at the end of the track about 100 meters away. A subject vehicle is parked at a starting point on the other end of the track. A simple drive maneuver is performed by the subject vehicle as follows: start from a stationary position, accelerate, stay at a constant speed for a few seconds, decelerate, and stop. The distance estimation algorithm records the detected vehicle's longitudinal distance estimation on a csv file along with a time stamp. Also, the ground truth data is produced from the vehicle CAN log. With the velocity log of the subject vehicle, the ground truth longitudinal distance from the target vehicle and time stamp are collected. Then, modal errors with 0.1 second interval and average error are calculated by comparing the algorithm estimation and the ground truth data.

3.5.4. Results

The results are shown in Table 1 of the Appendix section. The average error of the distance estimation is 8.77%. Figure 3.5.5 shows the algorithm estimation and ground

truth in a scatter plot. The distance estimation is very close to the ground truth up to about 20 meters, but the system starts losing the target car afterwards.

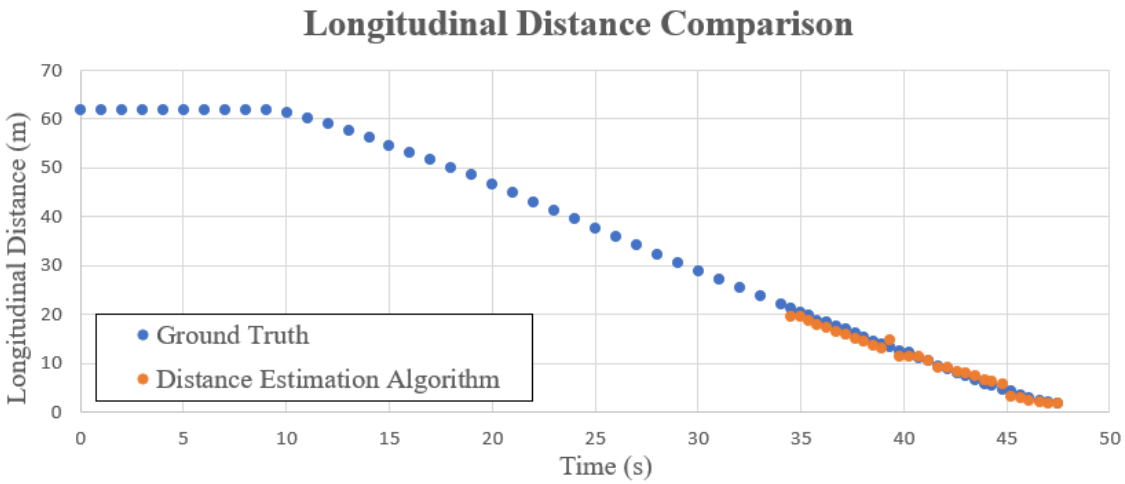

Figure 3.5.5. Distance Estimation and Ground Truth Comparison on a Scatter Plot

CHAPTER 4

CONCLUSIONS AND FUTURE WORK

4.1 Conclusions

This thesis introduces some of most popular ADAS techniques along with the implemented hardware and software architectures. The hardware architecture includes a camera and LIDAR to collect perception data about the surrounding environment and two single board computers to process the collected data. The computers run the software algorithms including lane mark detection (LMD), vehicle detection, vehicle tracking, longitudinal and lateral distance estimation, and sensor fusion. The outputs of these software algorithms can be utilized by the vehicle controller for any type of ADAS applications such as auto steering and adaptive cruise control.

4.1.1. Conclusion - Lane Mark Detection (LMD)

LMD development and results are shared in Section 3.2. The result is useful in checking whether a subject vehicle and detected vehicle ahead are driving in the host lane. As shown in Figure 3.2.16, on a straight or wide curve road, LMD shows a good result unless the pavement is not uniform, or the lane marks are hardly visible. LMD shows some failure when the subject vehicle is driving on an extreme curve or changing lanes. LMD sometimes falsely detect an outline of some objects as lane marks.

4.1.2. Conclusion - Vehicle Detection

Vehicle detection development and results are shared in Section 3.3. The algorithm detects the target vehicle most of the time when they are close to the subject vehicle. Most of the false negatives occur when the target vehicle is far away, and its size becomes small in the image. The precision of 95.82 % shows that the algorithm rarely output false positives, which occurs when the system detects oncoming vehicles. The algorithm fails sometimes when the orientation of the target vehicle tilts at an angle on the side of the image.

4.1.3. Conclusion - Vehicle Tracking

Vehicle tracking development and results are shared in Section 3.4. The vehicle tracking is combined with the vehicle detection and provides multi-vehicle tracking by assigning a track to each detected target. Using Kalman filter's prediction, the detected vehicle can still be tracked even when it is completely occluded temporally by some object. The Hungarian algorithm allows tracking of multiple vehicles even when they briefly overlap with each other and then break apart.

4.1.4. Conclusion - Longitudinal and Lateral Distance Estimation and Sensor Fusion

Longitudinal and lateral distance estimation and sensor fusion development and results are shared in Section 3.5. The camera-based distance estimation is not very reliable. But, when it is combined with the LIDAR distance measurement by the sensor fusion algorithm, the final longitudinal distance output is much more accurate with an average error of 8.77%. The maximum longitudinal range is about 20 meters due to the

limitation of the vehicle detection algorithm. If the detection algorithm is improved and detects vehicles that are farther away, the maximum longitudinal range can also be improved.

4.2 Future work

This thesis only covers a basic level of some ADAS techniques. ADAS is a technology that is growing at an extreme speed, and a large number of new papers and new algorithms are being developed and published every day. The proposed software algorithms can be improved in many ways and some possible ideas are shared.

For LMD, machine learning can also be used to detect lane marks.

For vehicle detection, YOLO version 3 is now available. It is faster and more accurate than YOLOv2 [19]. Also, the custom dataset can be expanded to include more variations and number of images. The training parameters used for training the detector can also be set differently to produce better results.

For vehicle tracking, more advanced Kalman filters such as Extended Kalman filter and Unscented Kalman filter can be used to handle non-linearity of the system.

For sensor fusion, as introduced in SSection 3.5.2, different levels of the sensor fusion model can be tried. The Kalman filter can also be used for sensor fusion.

APPENDIX

Table 1. Longitudinal Distance Estimation Result

Longitudinal Distance Comparison				
Time (s)	Ground Truth Distance (m)	Distance Estimation (m)	Modal Error (%)	Average Error (%)
34.5	21.21	19.64	7.44	8.77
35.0	20.33	19.43	4.47	
35.4	19.74	18.68	5.38	
35.8	18.65	17.91	3.99	
36.3	18.27	17.19	5.92	
36.7	17.48	16.50	5.62	
37.2	17.01	15.76	7.36	
37.6	16.03	15.09	5.88	
38.1	15.41	14.34	6.95	
38.6	14.58	13.63	6.54	
38.9	13.82	12.92	6.52	
39.4	13.26	14.61	10.12	
39.8	12.54	11.44	8.77	
40.3	12.15	11.48	5.54	
40.8	10.96	11.30	3.06	
41.2	10.51	10.44	0.65	
41.7	9.41	9.14	2.85	
42.1	8.87	9.02	1.61	
42.6	7.96	8.36	5.01	
43.0	7.26	7.90	8.79	
43.5	6.56	7.33	11.76	
44.0	5.71	6.57	15.03	
44.3	5.30	6.28	18.45	
44.8	4.57	5.67	24.16	
45.2	4.24	3.12	26.42	
45.7	3.35	2.81	16.17	
46.1	2.95	2.32	21.19	
46.6	2.33	2.07	11.35	
47.1	1.98	1.86	6.02	
47.5	1.78	1.78	0.07	

REFERENCES

[1] Lucintel, *Growth Opportunities in the Global Automotive ADAS Market*, https://www.researchandmarkets.com/research/2wcptx/growth.

[2] NVIDIA, *NVIDIA Jetson Modules and Developer Kits for Embedded Systems Development*, https://www.nvidia.com/en-us/autonomous-machines/embedded-systems-dev-kits-modules.

[3] NXP, *S32V234: Vision Processor for Front and Surround View Camera, Machine Learning and Sensor Fusion Applications*, https://www.nxp.com.

[4] OpenCV, *Color conversions*, https://docs.opencv.org/3.1.0/de/d25/imgproc_color_conversions.html.

[5] OpenCV, *Image Filtering*, https://docs.opencv.org/3.4/d4/d86/group__imgproc__filter.html#gaabe8c836e97159a9193fb0b11ac52cf1.

[6] C. Christopher, *Understanding Convolutions*, http://colah.github.io/posts/2014-07-Understanding-Convolutions.

[7] OpenCV, *Canny Edge Detection*, https://docs.opencv.org/3.4/da/d22/tutorial_py_canny.html.

[8] OpenCV, *Hough Line Transform*, https://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/hough_lines/hough_lines.html.

[9] S. Ren, K. He, R. Girshick, and J. Sun, “Faster R-CNN: Towards Real-time Object Detection with Region Proposal Networks,” *Advances in neural information processing systems*, pp. 91–99, 2015.

[10] J. Dai, Y. Li, K. He, J. Sun, “R-FCN: Object Detection via Region-based Fully Convolutional Networks,” in *NIPS*, 2016.

[11] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, “SSD: Single shot multibox detector,” European Conference on Computer Vision, pp. 21–37, 2016.

[12] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You Only Look Once: Unified, Real-time Object Detection,” in *CVPR*, 2016.

[13] J. Redmon, A. Farhadi, “YOLO9000: Better, Faster, Stronger,” in *CVPR*, 2017.

[14] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft, “COCO: Common objects in context,” in *ECCV*, 2014.

[15] R. Kalman, “A New Approach to Linear Filtering and Prediction Problems,” *Journal of Basic Engineering*, 1960(82), pp. 35-45.

[16] H. Patel, D. Thakore, “Moving Object Tracking Using Kalman Filter,” in *IJCSMC*, Vol. 2, Issue. 4, April 2013, pg. 326-332.

[17] X.Chen, X. Wang, J. Xuan, “Tracking Multiple Moving Objects Using Unscented Kalman Filtering Techniques,” *arXiv:1802.01235*, 2018.

[18] J. Llinas, C. Bowman, G. Rogova, A. Steinberg, E. Waltz and F. White, "Revisiting the JDL data fusion model II," in *Proceedings of the Seventh International Conference on Information Fusion*, 2004.

[19] J. Redmon, A. Farhadi, "YOLOv3: An Incremental Improvement," *arXiv:1804.02767,* 2018.